Lab Introduction

This lab gives a review of string, list, and dictionary methods. Work with a partner to complete the lab activity. Write your program in a file called crypto.py.

Lab Task

Review

This lab will utilize string, lists, and dictionaries in the activity. Included below are the previous labs to reference for each type of data structure. Make sure to review the provided lists of methods for each data type in each lab!

- Strings: Lab 05
- Lists: Lab 08
- Dictionaries: Lab 09

Cryptography

Cryptography is the art of hiding the meaning of a message in a way that the intended recipient can understand, but not anyone else. Fully secure cryptography requires a lot of detail-oriented nuance of implementation and some abstract algebra, but we can get some casual-level cryptography using what we know in Python so far.

You'll write several functions in crypto.py that are suitable for casual encryption. Feel free to use them to make messages harder for people who don't know the encryption to figure out, but don't use them for truly sensitive information; every encryption on this page can be broken…

Substitution Ciphers

In this lab you'll create functions for doing (and undoing) algorithms that belong to a class of encryption known as "substitution ciphers." These encrypt by taking the input text (called the "plain text") in small chunks (in our case by a single character) and replacing each chunk with a new chunk to occupy the same position in the output text (called the "cipher text"). The inverse functions generally decrypt via in the same chunk-by-chunk process. Your function will include a "key", some extra input that controls how it encrypts.

Shift Cipher

Code up the following cipher. You should write two functions: one to encrypt, encrypt_shift, and one to decrypt, decrypt_shift. It should always be the case that
`decrypt_shift(encrypt_shift(text, key), key) == text`

One of the first documented ciphers was the Caesar Cipher, which adds 3 to each letter (“a” becomes “d”, “b” becomes “e”, and so on). We will generalize that to add an arbitrary integer key, not just 3.

Your function encrypt_shift should take a message to encrypt and an integer key. The function should add the key to every letter in text, wrapping around, so e.g. a + 1 = b and z + 1 = a. Only change letters; leave non-letter characters as they are.

Using a similar method, implement the decrypt_shift function. This function will unscramble cipher text back into its original form. decrypt_shift takes in a string text variable and an integer key value.

Examples:
`encrypt_shift("caesar cipher", 3)` returns `"fdhvdu flskhu"`
`encrypt_shift("secret", 9)` returns `"bnlanc"`
`decrypt_shift("yk yqeemsq", 12)` returns `"my message"`

Hints

- You might want to try `'abcdefghijklmnopqrstuvwxyz'.find(character)` to convert characters to numbers
- Recall that find returns -1 if it does not find the character, so you’ll need to handle that specially for spaces and such to work
- You’ll probably want to use % to deal with wrap around values, i.e. z becomes c if you are shifting by 3

Password Cracking

Write a function called crack_passwords that takes in one parameter, string of multiple user login information, and returns a dictionary with the usernames as keys and corresponding decrypted passwords as the values. Your function should parse through the string parameter which will have the following format.

```
'''
username1, encrypted_password1, [mm]/[dd]
username2, encrypted_password2, [mm]/[dd]
username3, encrypted_password3, [mm]/[dd]
...
```

```
'''
```

The month and date represent the user's birthday. Assume that each user uses the day of their birthday as their encryption key (for example for the birthday of 3/10 use 10 as the decryption key). For example, for the following code should print:

```
user_logins1 = '''
Amir, rkzziqyvemui, 3/10
Bianca, fslmjwdgnwj, 6/18
Carlos, fssoasvq, 12/4
Diego, grmfqbojxop, 11/23
Elena, gdfwbuhwas, 5/14
'''
user_logins2 = '''
Felix, xjhwjylfwijs, 7/5
Gabe, gwzjsfaccb, 9/14
Hana, giohnuchxyq, 3/20
'''
cracked_pass1 = crack_passwords(user_logins1)
print(cracked_pass1)
cracked_pass2 = crack_passwords(user_logins2)
print(cracked_pass2)
```

```
{'Amir': 'happygolucky', 'Bianca': 'naturelover', 'Carlos': 'bookworm', 'Diego': 'jupitermars', 'Elena': 'springtime'}
{'Felix': 'secretgarden', 'Gabe': 'silvermoon', 'Hana': 'mountaindew'}
```

Hints

- The strip and split string methods will be helpful for this function
- Don't rewrite code you already have, once you have found the key, call decrypt_shift

Real Security

Current encryption relies on several things we haven't discussed here:

- Encrypting large blocks of text at once, instead of letter-by-letter
- Using methods of encryption based on abstract algebra and number theory which computer scientists are reasonably confident cannot be easily reversed
- Implementations that ensure each encryption of a block takes exactly the same amount of time, so that you can't infer things about the message from the timing
- The use of techniques beyond encryption, such as hashing (a topic you'll learn in another programming course), to establish other kinds of trust

However, one lesson learned from the Union's breaking of the Confederacy's use of Vignère ciphers in the US Civil War still holds today: the Union didn't know how to break the cipher quickly in general, but once it discovered that the Confederacy used three keys for almost all of its communications ("ManchesterBluff", "CompleteVictory", and "ComeRetribution") they were able to break the codes with ease. This remains true today: if you re-use passwords, or use passwords others might guess, good encryption doesn't help.

Conceptual Questions

When checking you off, a TA will ask you one of the following questions. Discuss the following questions with your partner as you are working through the lab.

- How did you convert the alphabet characters to numbers (without needing 26 if statements)?
- How does the strip() method work?
- How does the split() method work?